AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Currently amended) A method for using a hash table that is fully dynamic and lock-free, comprising:

performing a lookup into the hash table, wherein the lookup involves,

using a hash key to lookup a bucket pointer in a bucket array,

following the bucket pointer to a data node within a linked list containing all of the data nodes in the hash table, and

searching from the data node through the linked list to locate a node that matches the hash key if one exists;

wherein the linked list contains only data nodes and at most a constant number of dummy nodes; and

if the average number of data nodes in each bucket exceeds a maximum value:

increasing the number of buckets in the bucket array to form a larger bucket array, and

using more bits from the hash key to perform lookups in the larger bucket array,

wherein the data nodes are stored in the linked list in bit-inverted hash key order, and

wherein increasing the number of buckets in the bucket array involves mapping the existing bucket array into the top half of the larger bucket array.

2. (Original) The method of claim 1, wherein the data node pointed to by the bucket pointer precedes the nodes in the bucket.

3. (Original) The method of claim 1, wherein deleting the data node from the linked list involves:

using an atomic operation to mark the data node as dead; and

atomically updating the next pointer of the predecessor of the data node to point around the data node to the successor of the data node in the linked list.

4. (Currently amended) The method of claim 3 ~~claim 2~~, wherein deleting the data node from the linked list additionally involves redirecting the next pointer of the data node to become a back pointer that points to the predecessor of the data node.

5. (Original) The method of claim 4, wherein if a search through a chain of nodes from the back pointer does not lead to a live node, the method further comprises:

obtaining a parent bucket pointer;

searching through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and

updating the bucket pointer to point to the starting node.

6. (Original) The method of claim 2, wherein deleting the data node from the linked list involves using garbage collection or a solution to the repeat offender problem to reclaim the data node if possible.

7. (Original) The method of claim 1, further comprising generating the hash key by performing a pre-hashing operation to achieve a uniform distribution of hash keys over possible hash key values.

8 (Canceled).

9. (Currently amended) The method of claim 1 ~~claim 8~~, wherein buckets in the larger bucket array are initialized on-the-fly as they are referenced.

10. (Currently amended) The method of claim 1 ~~claim 8~~, wherein initializing a bucket pointer involves:

obtaining a parent bucket pointer for the bucket pointer;

searching through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and

updating the bucket pointer to point to the starting node.

11. (Original) The method of claim 1, wherein if there exists an old hash table, initializing a bucket pointer involves looking for a corresponding entry in the old hash table first, and if this fails:

obtaining a parent bucket pointer for the bucket pointer;

searching through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and

updating the bucket pointer to point to the starting node.

12 (Canceled).

13. (Currently amended) The method of claim 1 ~~claim 8~~,
wherein the data nodes are stored in the linked list in hash key order; and
wherein increasing the number of buckets in the bucket array involves interleaving the bucket array into the larger bucket array.

14. (Original) The method of claim 2, wherein if the average number of data nodes in each bucket falls below a minimum value, the method further comprises:
reducing the number of buckets in the bucket array to form a smaller bucket array; and
using fewer bits from the hash key to perform lookups in the smaller bucket array.

15. (Currently amended) A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for using a hash table that is fully dynamic and lock-free, the method comprising:
performing a lookup into the hash table, wherein the lookup involves,
using a hash key to lookup a bucket pointer in a bucket array,
following the bucket pointer to a data node within a linked list containing all of the data nodes in the hash table, and
searching from the data node through the linked list to locate a node that matches the hash key if one exists;
wherein the linked list contains only data nodes and at most a constant number of dummy nodes; and

if the average number of data nodes in each bucket exceeds a maximum value:

increasing the number of buckets in the bucket array to form a larger bucket array, and

using more bits from the hash key to perform lookups in the larger bucket array,

wherein the data nodes are stored in the linked list in bit-inverted hash key order, and

wherein increasing the number of buckets in the bucket array involves mapping the existing bucket array into the top half of the larger bucket array.

16. (Original) The computer-readable storage medium of claim 15, wherein the data node pointed to by the bucket pointer precedes the nodes in the bucket.

17. (Original) The computer-readable storage medium of claim 15, wherein deleting the data node from the linked list involves:

using an atomic operation to mark the data node as dead; and

atomically updating the next pointer of the predecessor of the data node to point around the data node to the successor of the data node in the linked list.

18. (Currently amended) The computer-readable storage medium of claim 17 ~~claim 16~~, wherein deleting the data node from the linked list additionally involves redirecting the next pointer of the data node to become a back pointer that points to the predecessor of the data node.

19. (Original) The computer-readable storage medium of claim 18, wherein if a search through a chain of nodes from the back pointer does not lead to a live node, the method further comprises:

obtaining a parent bucket pointer;

searching through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and

updating the bucket pointer to point to the starting node.

20. (Original) The computer-readable storage medium of claim 16, wherein deleting the data node from the linked list involves using garbage collection or a solution to the repeat offender problem to reclaim the data node if possible.

21. (Original) The computer-readable storage medium of claim 15, wherein the method further comprises generating the hash key by performing a pre-hashing operation to achieve a uniform distribution of hash keys over possible hash key values.

22 (Canceled).

23. (Currently amended) The computer-readable storage medium of claim 15 ~~claim 22~~, wherein buckets in the larger bucket array are initialized on-the-fly as they are referenced.

24. (Currently amended) The computer-readable storage medium of claim 15 ~~claim 22~~, wherein initializing a bucket pointer involves:

obtaining a parent bucket pointer for the bucket pointer;

searching through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and

updating the bucket pointer to point to the starting node.

25. (Original) The computer-readable storage medium of claim 15, wherein if there exists an old hash table, initializing a bucket pointer involves looking for a corresponding entry in the old hash table first, and if this fails:

obtaining a parent bucket pointer for the bucket pointer;

searching through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and

updating the bucket pointer to point to the starting node.

26 (Canceled).

27. (Currently amended) The computer-readable storage medium of claim 15 ~~claim 22~~,

wherein the data nodes are stored in the linked list in hash key order; and

wherein increasing the number of buckets in the bucket array involves interleaving the bucket array into the larger bucket array.

28. (Original) The computer-readable storage medium of claim 15, wherein if the average number of data nodes in each bucket falls below a minimum value, the method further comprises:

reducing the number of buckets in the bucket array to form a smaller bucket array; and

using less bits from the hash key to perform lookups in the smaller bucket array.

29. (Currently amended) An apparatus that implements a hash table that is fully dynamic and lock-free, comprising:

a lookup mechanism, wherein the lookup mechanism is configured to,

use a hash key to lookup a bucket pointer in a bucket array,

follow the bucket pointer to a data node within a linked list containing all of the data nodes in the hash table, and to

search from the data node through the linked list to locate a node that matches the hash key if one exists;

wherein the linked list contains only data nodes and at most a constant number of dummy nodes; and

a bucket array expansion mechanism, wherein if the average number of data nodes in each bucket exceeds a maximum value, the bucket array expansion mechanism is configured to:

increase the number of buckets in the bucket array to form a larger bucket array, and to

use additional bits from the hash key to perform lookups in the larger bucket array,

wherein the data nodes are stored in the linked list in bit-inverted hash key order, and

wherein the bucket array expansion mechanism is configured to map the existing bucket array into the top half of the larger bucket array.

30. (Original) The apparatus of claim 29, wherein the data node pointed to by the bucket pointer precedes the nodes in the bucket.

31. (Original) The apparatus of claim 29, further comprising a node deletion mechanism configured to:

use an atomic operation to mark the data node as dead; and to

atomically update the next pointer of the predecessor of the data node to point around the data node to the successor of the data node in the linked list.

32. (Currently amended) The apparatus of claim 31 ~~claim 30~~, wherein the node deletion mechanism is additionally configured to redirect the next pointer of the data node to become a back pointer that points to the predecessor of the data node.

33. (Original) The apparatus of claim 32, further comprising a bucket pointer updating mechanism, wherein if a search through a chain of nodes from the back pointer does not lead to a live node, the bucket pointer updating mechanism is configured to:

obtain a parent bucket pointer;

search through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and to

update the bucket pointer to point to the starting node.

34. (Original) The apparatus of claim 30, wherein the node deletion mechanism is additionally configured to use garbage collection or a solution to the repeat offender problem to reclaim the data node if possible.

35. (Original) The apparatus of claim 29, further comprising a pre-hashing mechanism configured to generate the hash key by performing a pre-hashing operation to achieve a uniform distribution of hash keys over possible hash key values.

36 (Canceled).

37. (Currently amended) The apparatus of claim 29 ~~claim 36~~, further comprising a bucket initialization mechanism configured to initialize buckets in the larger bucket array on-the-fly as they are referenced.

38. (Currently amended) The apparatus of claim 29 ~~claim 36~~, wherein the bucket initialization mechanism is configured to:

obtain a parent bucket pointer for the bucket pointer;

search through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and to

update the bucket pointer to point to the starting node.

39. (Original) The apparatus of claim 29, wherein if there exists an old hash table, the bucket initialization mechanism is configured to look for a corresponding entry in the old hash table first, and if this fails to:

obtain a parent bucket pointer for the bucket pointer;

search through the linked list from a node pointed to by the parent bucket pointer to locate a starting node for the bucket pointer; and to

update the bucket pointer to point to the starting node.

40 (Canceled).

41. (Currently amended) The apparatus of claim 29 ~~claim 36~~,

wherein the data nodes are stored in the linked list in hash key order; and

wherein the bucket array expansion mechanism is configured to interleave the bucket array into the larger bucket array.

42. (Original) The apparatus of claim 30, further comprising a bucket array contraction mechanism, wherein if the average number of data nodes in each

bucket falls below a minimum value, the bucket array contraction mechanism is configured to:

reduce the number of buckets in the bucket array to form a smaller bucket array; and to

use less bits from the hash key to perform lookups in the smaller bucket array.